02. 순방향 신경망

순방향 신경망 (FNN)

FNN : Feedforward Neural Network (or Fully Connected Neural Network)

엄밀히 다른 개념이지만 대부분의 FNN이 fully connected 구조를 쓰기 때문에 혼용해서 씀

다층 퍼셉트론의 다른 이름, 인공 신경망 모델 중 가장 기본이 되는 모델

범용 근사 정리(universal approximation theorem)을 통해 n차원 공간의 연속 함수를 근사할 수 있음.

2.1 순방향 신경망의 구조와 설계 항목

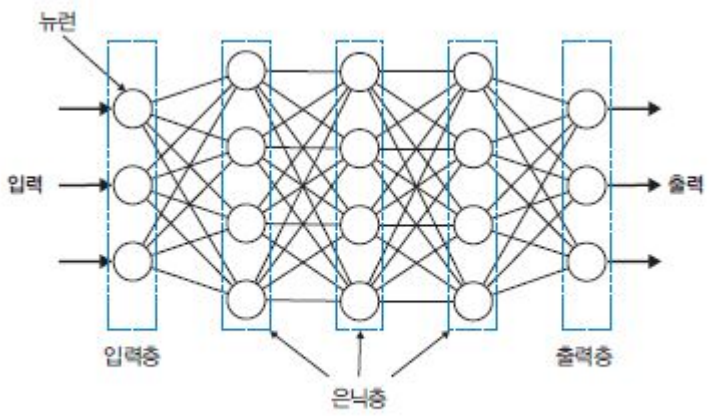
▼ 정리

FNN은 데이터가 한 방향으로 전달되는 순방향 연결 만을 가지는 구조로 되어 있음.

퍼셉트론의 연산과 같은 기본 뉴런 연산으로 실행됨.

순방향 신경망은 데이터 구조에 대한 특별한 가정 사항 x → 데이터는 서로 독립되어 있다고 가정함.

계층 구조



▶ 그림 1 인공 신경망 구조

입력 계층, 은닉 계층, 출력 계층으로 구분됨

- 입력 계층 : 외부에서 데이터 전달받음
- 은닉 계층 : 데이터의 특징 추출
- 출력 계층 : 추출된 특징을 기반으로 추론한 결과를 외부에 출력

입력, 출력 계층은 하나씩 있지만 은닉 계층의 경우 문제의 복잡도에 따라 가변적으로 구성

완전 연결 계층

순방향 신경망은 모든 계층이 fully connected layer로 구성. → 계층에 속한 각 뉴런이 이전 계층의 모든 뉴런과 모두 연결된 구조

같은 입력 데이터에서 뉴런마다 서로 다른 특징을 추출함

→ 데이터에 특징이 많을수록 그에 비례해 **뉴런 수가 충분히 많아야** 데이터의 특징을 모두 추출 가능함.

뉴런은 데이터에 내재한 특징 추출을 위해 가중 합산, 활성 함수를 순차적으로 실행

- 가중 합산 : 추출할 특징에 중요한 영향을 미치는 데이터를 선택하는 과정
  - 입력값 각각에 가중치 곱해서 모두 더한 것

$$z = \sum_{i=1}^{n} w_i \cdot x_i + b$$

  - 중요한 feature은 큰 가중치, 덜 중요한 feature은 작은 가중치 곱함

→ 입력의 영향력을 조절하여 학습

- 편향 b를 더하는 이유 : 특징을 공간상 임의의 위치에 표현하기 위해서
- 강화학습에서 가치 함수 근사할 때 사용하는 개념이랑 같음

- 활성 함수 : 원하는 형태로 특징을 추출하기 위해 데이터를 비선형적으로 변환하는 과정
  - 복잡한 패턴 학습하려면 비선형 함수는 필수 (XOR 문제 같은 건 선형 함수로 절대 불가능)

ReLU

: Rectified Linear Unit

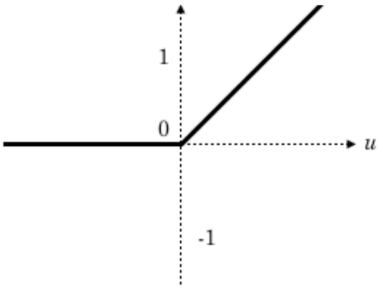
기본적인 활성 함수 (구간 선형 함수), 딥러닝에서 가장 많이 쓰임

- 입력값 > 0

  → 그대로 출력

- 입력값 ≤ 0

  → 0 출력



- 실함수

  : 입력은 실수/벡터 상관 없음, 출력도 실수인 함수

  - 뉴런은 실함수

- 벡터 함수

  : 입력도 벡터, 출력도 벡터인 함수

  - 계층은 벡터 함수
  - 신경망도 벡터 함수지만, 벡터 함수들의 합성 함수이기도 함.

$$f(x) = L_n(L_{n-1}(...L_1(x)))$$

신경망은 학습 시 미분을 사용하기 때문에 신경망이 표현하는 함수는 미분 가능 해야 함.

범용 근사 정리

: 2계층의 순방향 신경망에서 은닉 뉴런을 충분히 사용하고 검증된 활성 함수를 사용하면, n차원 공간의 임의의 연속 함수를 원하는 정도의 정확도로 근사할 수 있음.

→ 아주 복잡한 함수를 정확히 근사하려면 은닉 뉴런을 충분히 사용 해야 함

- 뉴런 수 많아짐 → 과적합 쉽게 발생 → 모델 성능 저하

  ⇒ 깊은 신경망으로 확장

  : 같은 성능의 모델을 적은 수의 뉴런을 사용해 만들 수 있음

순방향 신경망의 설계 항목

1. 모델의 입력 형태
2. 모델의 출력 형태
   - 출력 형태
   - 활성 함수
3. 은닉 계층
   - 활성 함수
4. 네트워크 크기
   - 네트워크 깊이
   - 네트워크 폭

2.2 분류와 회귀 문제

▼ 정리

- 분류 문제 : 범주형 데이터를 예측하는 문제
- 회귀 문제 : 숫자형 데이터를 예측하는 문제

분류 문제

: 데이터의 클래스 또는 카테고리를 예측하는 문제

- 이진 분류 : 두 개 클래스로 분류하는 문제

  ex) 스팸 메일 여부, 암 진단 결과 여부, 사진의 합성 여부 등

- 다중 분류 : 여러 클래스로 분류하는 문제

  ex) 강아지 품종 분류, 질병 진단, 표정 종류 등

- 판별 함수

  모델은 입력 데이터가 속한 클래스 예측

  만약 고양이 → 고양이 클래스로 예측함

- 확률 모델

  입력 데이터가 각 클래스에 속할 확률 에측

  만약 고양이 → 고양이, 개, 토끼 클래스에 속할 확률 동시 출력

  확률 모델은 표현하는 정보가 많음

  예측된 확률분포에서 샘플링 → 입력 데이터가 같아도 조금씩 다른 출력 만들 수 있음

회귀 문제

: 여러 독립 변수와 종속 변수의 관계를 연속 함수 형태로 분석하는 문제

- 예측값이 숫자형 데이터면 회귀 문제

  ex) 주가 및 경제 트렌드 예측, 상품 수요/공급량 예측, 주변에 보이는 사물의 위치/거리 추정, 교통량/주행 시간 예측 등

- 입력 데이터에 대한 함숫값을 예측함

분류 문제는 데이터의 클래스를 예측하는 문제로 두 클래스로 분류하면 이진 분류, 여러 클래스로 분류하면 다중 분류라고 한다. 분류 모델을 판별 함수로 정의하면 모델은 입력 데이터가 속한 클래스를 예측하지만, 확률 모델로 정의하면 입력 데이터가 각 클래스에 속할 확률을 예측한다.

회귀 문제는 여러 독립 변수와 종속 변수의 관계를 연속 함수 형태로 분석하는 문제이다. 회귀 모델은 입력 데이터에 대한 함숫값을 예측하는데, 확률 모델로 정의하면 관측값의 확률분포를 예측한다.

2.3 이진 분류 모델

▼ 정리

베르누이 분포를 예측하는 모델로 이진 분류 모델 정의 가능함

$$P(X=x)=p^x(1-p)^{1-x}$$

p : 1이 나올 확률 (성공 확률)

p - 1 : 0이 나올 확률 (실패 확률)

i.i.d = independent and identically distributed

: 독립이면서 동일 분포 (서로의 결과에 영향이 없고, 같은 확률 분포를 따름)

출력 계층의 활성 함수

폐의 엑스레이 사진으로 코로나 감염 여부를 판별하는 이진 분류 모델을 만든다고 가정.

모델은 폐의 엑스레이 사진을 입력받아 코로나에 감염되었을 확률 p를 예측함.

이때 신경망 모델은 코로나에 감염되었을 수치를 점수(score) 또는 로짓(logit)으로 예측하고, 이를 활성 함수를 통해 베르누이 확률분포의 파라미터 p로 변환함.

→ 이때 사용하는 활성 함수 = 시그모이드 함수(sigmoid)

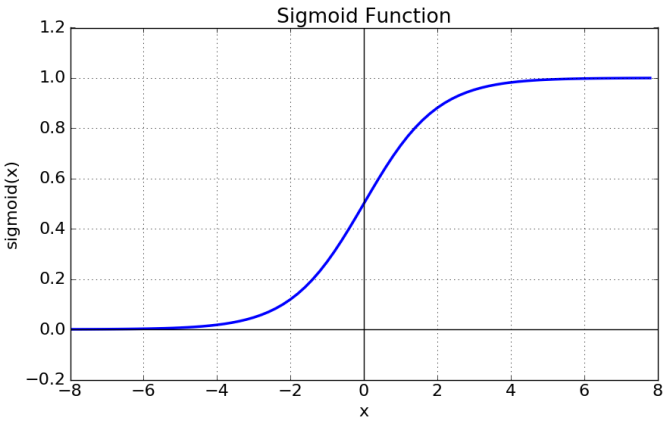
출력 계층에서 시그모이드 함수 사용 시 손실 함수를 크로스 엔트로피로 선택해야 학습 안정화

시그모이드(Sigmoid) 함수

: 신경망의 출력 계층에서 실숫값을 확률로 변환할 때 사용함.

값을 [0, 1] 범위로 만들어 주기 때문에 값을 고정 범위로 변환하는 스쿼싱(squashing) 함수로 사용됨.

- squashing 함수 : 입력이 크든 작든 출력을 일정한 범위로 제한하는 함수



$$\sigma(x) = \frac{1}{1+e^{-x}}$$

→ 로지스틱 함수

- 로짓 함수는 확률을 log-odd로 바꾸는 함수이고, 확률적 출력을 선형으로 표현할 때 사용
  - 로짓과 로지스틱 함수는 서로 역함수 관계. ⇒ 로지스틱 함수의 입력 = 로짓

이진 분류 문제는 동전 던지기를 할 때 앞면과 뒷면이 나올 확률을 예측하는 문제와 같다.

이진 분류 모델은 베르누이 분포를 예측하는 모델로 정의할 수 있다.

2.4 다중 분류 모델

▼ 정리

다중 분류 모델은 카테고리 분포(다항 분포 1회 시행)을 예측하는 모델로 정의 가능

카테고리 분포

: 베르누이 분포의 다항 확장판. 여러 종류의 사건이 발생할 확률을 나타냄. K개 사건의 확률을 표현

$$P(X = i) = p_i, \quad \text{where } \sum_{i=1}^{K} p_i = 1$$

출력 계층의 활성 함수

고양이 사진을 입력받아서 분류하는 다중 분류 모델을 만든다고 가정.

모델은 고양이 사진을 입력받아서 다람쥐, 고양이, 강아지, ... 와 같은 K개 클래스에 속할 확률인 p를 예측함.

이 때 신경망 모델은 각 클래스에 속할 수치인 점수 또는 로짓을 실수 벡터로 출력, 활성 함수를 통해 카테고리 분포의 확률 벡터 p로 변환함.

→ 이 때 사용하는 활성 함수 = 소프트맥스(softmax) 함수

이진 분류 문제도 다중 분류 문제로 풀 수 있음.

- 이진 분류 문제의 출력 : 실수
- 다중 분류 문제의 출력 : 크기가 K인 실수 벡터

소프트맥스 함수도 시그모이드와 마찬가지로 손실 함수를 크로스 엔트로피로 선택해야 학습이 안정화됨.

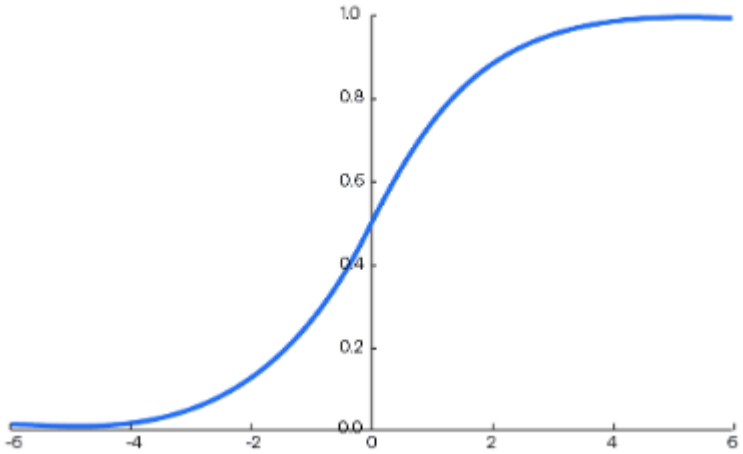
소프트맥스(Softmax) 함수

: 실수 벡터를 확률 벡터로 변환함.

실수 벡터의 각 요소는 [0, 1] 범위로 변환 → 각 요소의 합은 1

입력값 클수록 1에 가깝게, 작을수록 0에 가깝게 만들어 줌.

$$\text{Softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}$$



- 소프트맥스는 시그모이드를 여러 클래스에 대해 일반화한 함수.

  → 소프트맥스로 계산한 각 클래스의 확률은 시그모이드 함수와 같은 모양이 됨.

Softmax는 입력 벡터를 확률 분포로 바꾸는 함수이다. (지수를 써서 상대적 크기를 강조함)

2.5 회귀 모델

▼ 정리

회귀 문제

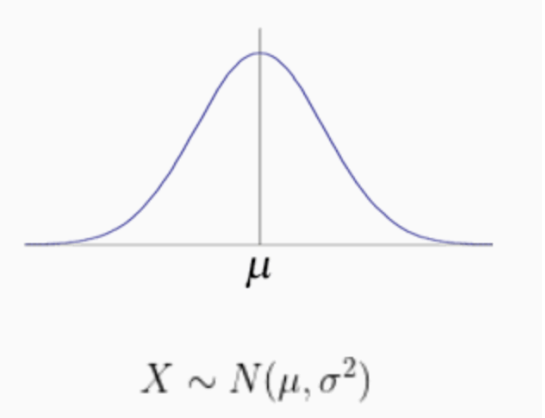
: 여러 독립 변수와 종속 변수의 관계를 연속 함수 형태로 분석하는 문제

관측 오차 / 실험 오차는 **가우시안 분포**(정규분포)로 정의됨.

→ 회귀 문제는 가우시안 분포를 예측하는 모델로 정의 가능

가우시안 분포

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$



중심 극한 정리에 의해 독립적인 확률 변수들의 평균은 가우시안 분포에 가까워지는 성질이 있음.

중심 극한 정리 : 어떤 분포를 따르든지 간에, i.i.d 확률 변수 여러 개의 평균을 취하면, 그 평균은 정규분포에 가까워진다.

회귀 모델 정의

관측 데이터는 N개의 (xi, ti)로 구성

$$\mathcal{D} = \{(x_i, t_i)\}_{i=1}^{N}$$

입력 데이터 x가 같은 분포에서 독립적으로 샘플링되어 i.i.d를 만족한다고 가정함.

타깃 t는 모델 예측값 y에 관측 오차가 더해진 값으로 정의됨.

$$t_i = y(x_i) + \epsilon_i$$

이 때 관측 오차는 가우시안 분포를 따름.

→ 오차의 분산은 정밀도의 역수 → 상수로 가정

분산이 크면 정밀하지 않고, 분산이 작으면 정밀한 것으로 생각할 수 있음.

$$\beta = \frac{1}{\sigma^2}$$

$$p(t_i|x_i) = \mathcal{N}(t_i|y(x_i), \beta^{-1})$$

→ 타깃 t의 확률분포

타깃 t의 분포는 관측 오차의 분산을 갖는 가우시안 분포로 정의됨

→ 그래서 신경망 모델은 평균 y(xi)만 예측하면 됨

⇒ 가우시안 분포는 **평균**만 알면 중심이 어딘지 정확히 알 수 있는 분포이기 때문

이 개념에서 관측 오차는 통제할 수 없기 때문에 그나마 가장 중심의 평균값만 예측하는 게 최선!

관측 오차가 가우시안 분포를 따른다 = 정답이 평균을 중심으로 퍼져 있다

그렇다면 최선은? 그 중심(평균)을 맞추는 것

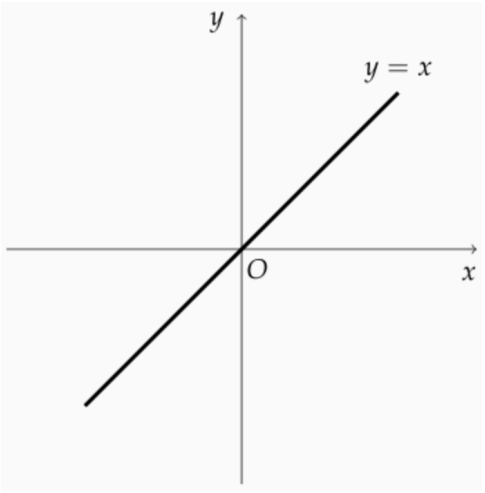
⇒ 신경망은 평균만 예측해도 충분함

- 관측 오차
  - 실제 정답(t)이 우리가 생각하는 모델 출력(y)과 다르게 나오는 이유를 설명하는 노이즈
  - ex) 센서 오차, 인간의 편향, 환경 요인 등
  - 역할 : 신경망/회귀 모델이 학습할 때 오차 구조를 반영하여 평균을 예측하게 만듦

출력 계층의 활성 함수

집값을 결정하는 요인으로 구성된 '방의 개수, 면적, 집 종류, 역과의 거리' 데이터를 입력받아 집값을 예측하는 회귀 모델을 만든다고 가정.

모델은 데이터 입력받아 집값의 가우시안 분포의 평균을 예측함



회귀 모델은 예측된 평균/분산이 바뀌면 안되기 때문에 항등함수 사용

→ 회귀 모델에서는 오직 평균만 학습하고 분산은 상수로 간주됨 → 평균값이 달라져도 분산은 일정

2.6 입력 계층

▼ 정리

입력 데이터를 벡터 형태로 받아서 다음 계층에 전달하는 역할

입력 데이터가 크기가 n인 벡터라면 입력 계층은 n개의 뉴런으로 정의됨

만약 28x28 크기의 필기체 숫자 이미지로 필기체 인식

→ 이미지를 1차원으로 변환한 크기가 784인 1차원 벡터를 입력받는 입력 계층을 구성함

2.7 활성 함수

▼ 정리

활성 함수

: 은닉 계층을 설계할 때 선택할 수 있음

- 시그모이드 계열

  : S자형 곡선 형태

- 시그모이드
- 하이퍼볼릭 탄젠트

→ 연산 속도가 느리고 그레이디언트 소실의 원인이 되어 신경망 학습에는 좋지 않음

→ 하지만 값을 고정 범위로 만들어 주는 스쿼싱 기능이 필요한 구조에서 다양하게 활용됨

그레이디언트 소실

: 역전파 중에 기울기가 점점 작아져서 앞단(입력층 쪽)까지 거의 전달되지 않는 현상

스쿼싱 기능

: 실숫값을 고정 범위로 만들어 주는 기능

- ReLU 계열

  : 구간 선형 함수

  - ReLU
  - 리키 ReLU
  - 맥스아웃
  - ELU

  → 선형성을 갖고 있어 연산 속도가 매우 빠르고 학습 과정을 안정적으로 만들어 줌

  → 따라서 은닉 계층에서는 ReLU 계열을 활성 함수로 사용하는 것이 좋음

계단 함수

: 매컬러-피츠 모델과 퍼셉트론에서 사용된 활성 함수. 생체 뉴런의 발화 방식을 모방해 만듦

뉴런의 활성과 비활성 상태를 숫자 1과 0으로 표현

→ 입력값이 0보다 크면 1을 출력 / 0보다 작으면 0을 출력

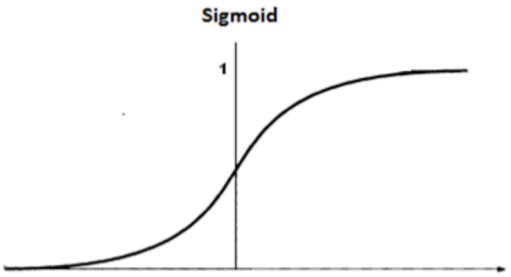
$$H(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$$

하지만 현대의 신경망에서는 사용할 수 없음

→ 미분을 이용해 학습하는 역전파 알고리즘에 적용 x → 시그모이드 함수 등장

시그모이드 계열

시그모이드 함수



$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

모든 구간에서 미분 가능. 증가 함수 이므로 미분값 항상 양수

함숫값 범위 [0, 1]

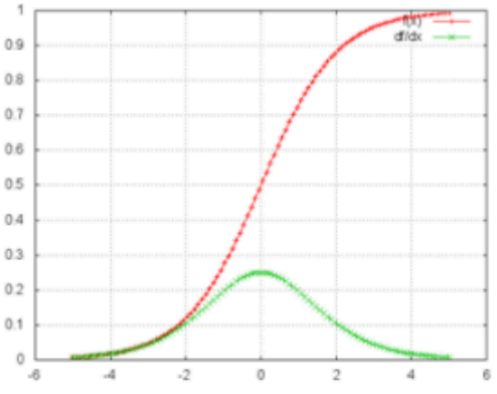
문제점

- 함수 정의에 지수 함수 포함 → 연산 비용 높음
- 그레이디언트 포화가 발생해서 학습이 중단될 수 있음
- 양수만 출력하므로 학습 경로가 진동하면서 학습 속도가 느려짐

그레이디언트 포화

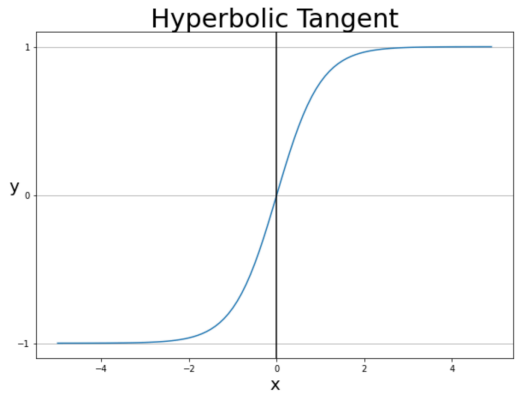
: 시그모이드 함수 끝부분에서 미분값이 0으로 포화되는 상태

- 포화 : 입력값이 변화해도 함숫값이 변화하지 않는 상태
- 그레이디언트가 0으로 포화되면 그레이디언트 손실이 일어나 학습 진행 안됨



그리고 시그모이드의 출력은 항상 양수라서 학습 시 최적화 경로가 최적해를 향해 곧바로 가지 못하고 좌우로 진동 → 비효율!!

하이퍼볼릭 탄젠트 함수



$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = 2\sigma(2x) - 1$$

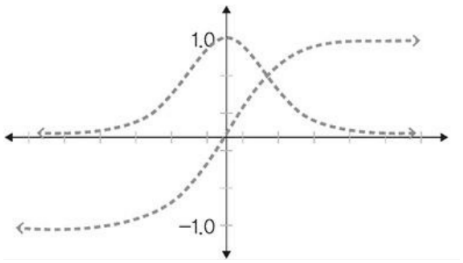
시그모이드가 항상 양수만을 출력하기 때문에 최적화가 비효율적인 문제를 해결하고자 사용됨

함숫값 범위 [-1, 1]

하지만 여전히 나머지 두 문제점은 해결되지 못했음

문제점

- 함수 정의에 지수 함수가 포함 → 연산 비용 높음
- 그레이디언트 포화가 발생해 학습 중단될 수 있음

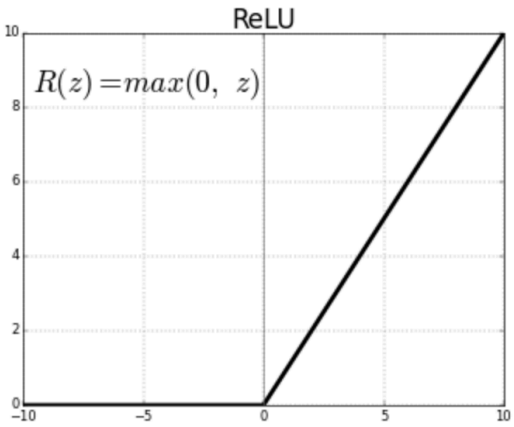


ReLU 계열

ReLU 함수

: 0보다 큰 입력이 들어오면 그대로 통과, 0보다 작은 입력이 들어오면 0 출력

→ 입력값이 양수인 경우에만 활성 상태가 됨



$$f(x) = \max(0, x)$$

계단 함수보다 좀 더 생체 뉴런에 가깝게 만들어짐.

ReLU를 사용하면 시그모이드 계열보다 추론과 학습 속도가 빨라지고 안정적으로 학습할 수 있음

→ 연산이 거의 없기 때문에 / 미분을 계산할 필요가 없기 때문에

⇒ 시그모이드 계열보다 6배 정도 학습이 빨라짐

ReLU는 양수 구간이 선형 함수이기 때문에 그레이디언트 소실이 생기지 않음 → 안정적인 학습

단점

- 양수만 출력하므로 학습 경로가 진동하면서 학습 속도가 느려짐
- 죽은 ReLU가 발생하면 학습이 진행되지 않을 수 있음

죽은 ReLU : 뉴런이 계속 0을 출력하는 상태

- 뉴런이 계속 0을 출력하게 되면 그레이디언트도 0이 되어 뉴런이 더 학습되지 않고 같은 값을 출력하게 됨
- 가중치 초기화를 잘못했거나 학습률이 매우 클 때 발생 가능
- 뉴런의 10-20%가 죽은 ReLU가 되면 학습에 문제가 될 수 있음

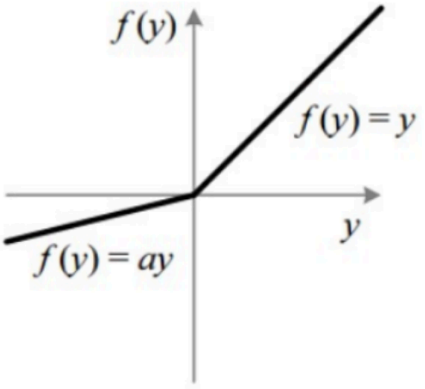
릭키 ReLU, PReLU, ELU 함수

죽은 ReLU 문제 해결 방안 : 음수 구간이 0이 되지 않도록 약간의 기울기 주기

⇒ 릭키 ReLU

- 기울기 고정이므로 최적 성능 보장 x

$$f(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha x & \text{if } x < 0 \end{cases} \quad (\alpha \ , \ 0.01)$$



⇒ PReLU

- 기울기 학습 가능
- 뉴런별로 기울기 학습하므로 성능 개선
- 릭키 ReLU 나 PReLU 는 음수 구간이 직선이므로 기울기가 작더라도 큰 음숫값이 들어오면 출력값이 -무한대로 발산 가능

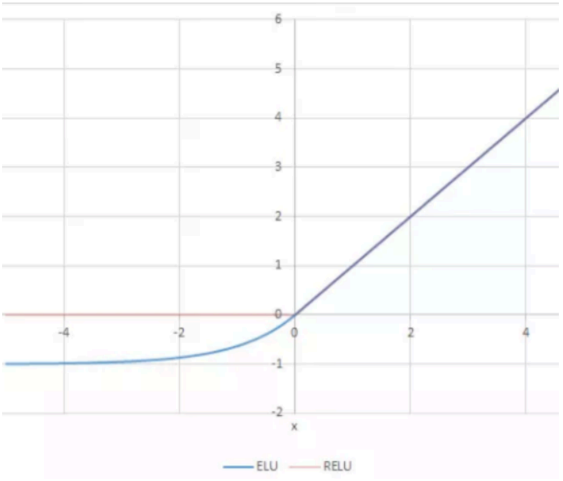
$$\text{Leaky ReLU(x)} = \begin{cases} x & if\ x > 0 \\ 0.01x & otherwise \end{cases}$$

$$\text{PReLU(x)} = \begin{cases} x & if\ x > 0 \\ ax & otherwise \end{cases}$$

⇒ ELU

- 음수 구간이 지수 형태이므로 x값이 음수 방향으로 커지더라도 함숫값이 0에 가까운 일정한 음숫값으로 포화됨
- 아주 큰 음숫값이 들어와도 노이즈에 민감해지지 않음

$$f(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha(e^x - 1) & \text{if } x < 0 \end{cases} \quad (\alpha > 0)$$



맥스아웃 함수

: 활성 함수를 구간 선형 함수로 가정하고, 각 뉴런에 최적화된 활설 함수를 학습을 통해 찾아냄

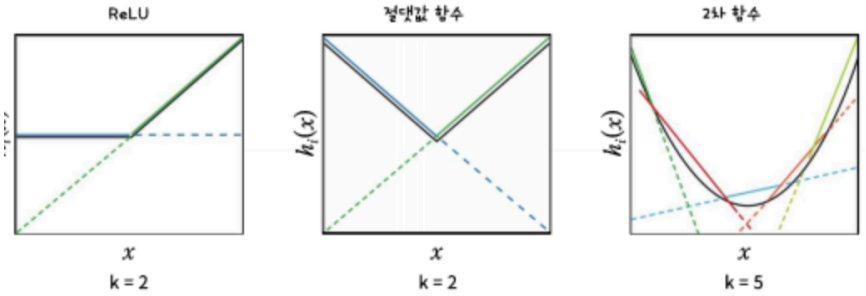
$$f(x) = \max(w_1^\top x + b_1, w_2^\top x + b_2)$$

- ReLU 의 일반화된 형태
- 뉴런별로 선형 함수를 여러 개 학습한 뒤에 최댓값 취함
- 성능이 뛰어남
- 맥스아웃 유닛 구조
  - 맥스아웃 활성 함수를 학습하기 위해 뉴런을 확장한 구조
  - 선형 노드, 최댓값을 출력하는 노드로 구성
    - 선형 노드 : 뉴런의 가중 합산과 같은 형태로 선형 함수 학습
    - 최댓값 출력 노드 : 구간별로 최댓값을 갖는 선형 함수 선택
- 볼록 함수 근사 능력
  - 맥스아웃은 선형 노드의 개수에 따라 다른 형태의 볼록 함수를 근사할 수 있음

- 선형 노드 2개 → ReLU, 절댓값 함수 근사
- 선형 노드 5개 → 2차 함수 근사
- 선형 노드가 많아지면 좀 더 부드러운 곡선 형태의 활성 함수를 학습할 수 있음

  → 너무 많아지면 맥스아웃 유닛에 사용되는 파라미터 개수도 비례해서 늘어남

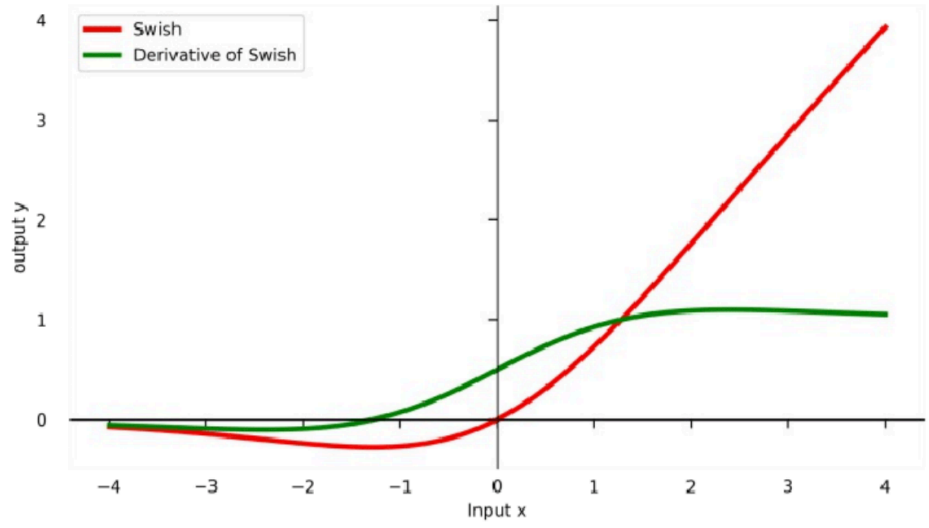
  → 선형 노드를 제한적으로 늘려줘야 함



⇒ 맥스아웃 적용하면 유닛별로 파라미터 수는 증가하지만, 전체 신경망의 깊이 자체가 줄어들어서 전체 파라미터 개수는 소폭 증가!!

Swish 함수

$$\text{Swish}(x) = x \cdot \sigma(x) = x \cdot \frac{1}{1+e^{-x}}$$



구글에서 제안한 활성 함수. SiLU로 불리기도 함

- 선형 함수 x와 시그모이드의 곱으로 정의됨
- 생체 뉴런의 발화 방식에 가장 가까우면서도 인공 신경망의 성능에 최적인 활성 함수 형태로 추정
- SELU, GELU 와 같은 ReLU 계열의 활성화 함수들이 Swish 함수와 유사함

2.8 신경망 모델의 크기

▼ 정리

신경망 모델의 크기 : 너비, 깊이

- 너비 : 계층별 뉴런 수
- 깊이 : 계층 수

데이터에 특징 많고 데이터 간의 관계 복잡 → 뉴런 수 증가

특징의 추상화 수준 높음 → 계층 수 증가

→ 가늠하기 어렵기 때문에 최적의 크기를 탐색해 나가야 함

모델 크기 탐색

- 그리드 서치

  : 파라미터별로 구간 정해 등간격으로 값을 샘플링
- 랜덤 서치

  : 여러 파라미터 조합해 랜덤하게 값을 샘플링

→ 랜덤 서치가 더 골고루 테스트 → 그리드보다 성능 좋음

최근에는 네트워크 구조 탐색 방법인 NAS 와 같은 자동 모델 탐색 방법 활용하기도 함

→ 강화 학습, 유전 알고리즘, 베이지안 기법 등으로 구현

모델 크기 조정

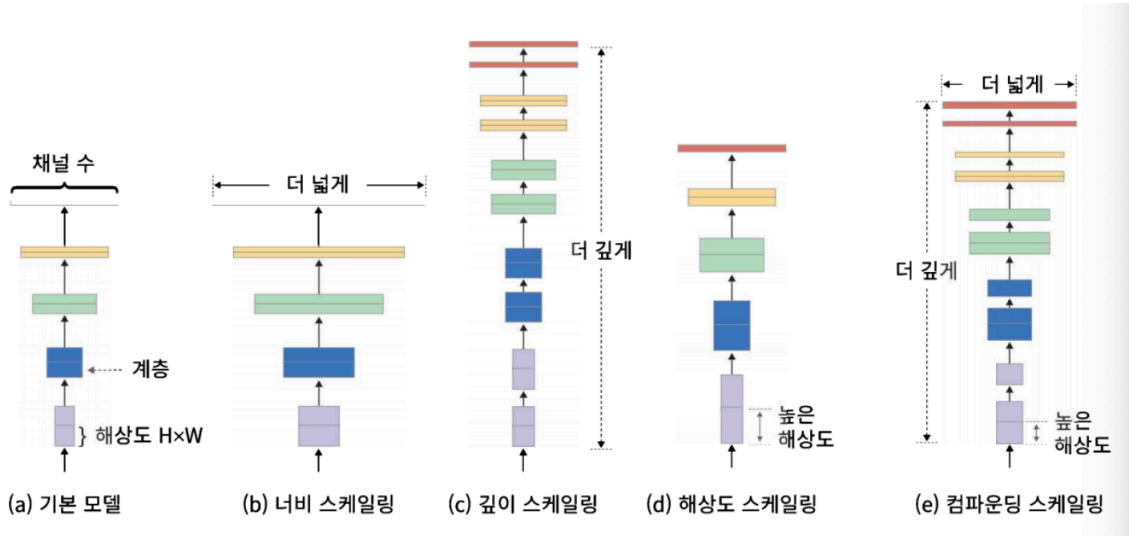
신경망 모델 구성 시 가장 많이 사용하는 방식

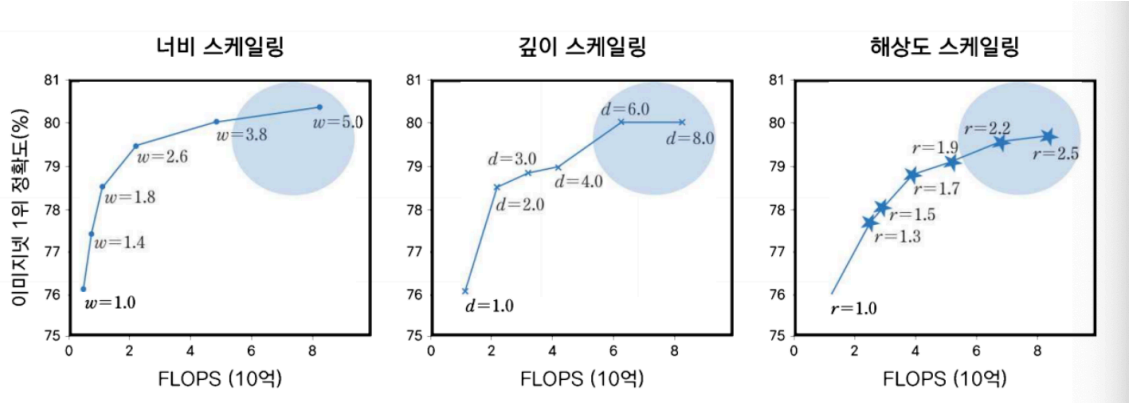
= 성능이 검증된 기본 모델을 새로운 문제에 맞게 크기 조정하는 방식

구글에서 제안한 모델인 이피션트넷(EfficientNet)은 너비, 깊이, 입력 이미지의 해상도 셋을 균형 있게 동시에 늘리는 방식이 가장 효율적이라는 것을 보였음.

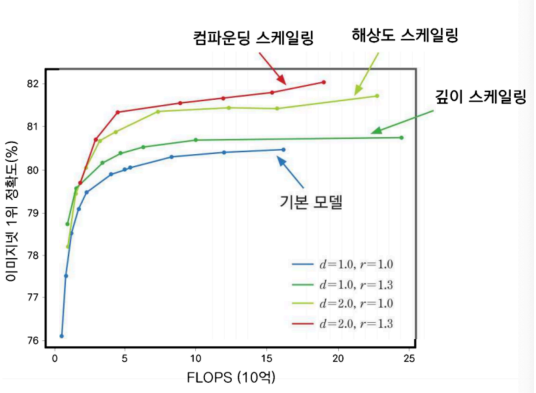
단순히 한쪽 방향으로 키우는 것보다, 세 가지 축을 비율에 맞게 동시에 확장하는 것이 더 높은 정확도, 더 적은 파라미터, 더 적은 연산량을 달성할 수 있음.

⇒ 컴파운드 스케일링 모델





각각 늘릴수록 성능이 향상되다 일정 수준을 넘어가면 포화 상태 혹은 성능이 더이상 개선되지 않는 모습을 볼 수 있음.



반면 기본 모델의 너비, 깊이, 입력 이미지의 해상도를 동시에 늘리면 다른 스케일링 방법과 비교해서 가장 좋은 성능을 보여줌.

⇒ 따라서 모델의 크기를 늘릴 때는 세가지 요소를 모두 고려해야 함

신경망 모델의 크기는 '계층의 수'를 나타내는 깊이와 '계층별 뉴런 수'를 나타내는 너비로 말할 수 있다.

신경망 모델의 크기를 정할 때는 그리드 서치나 랜덤 서치와 같은 탐색 방법을 사용한다. 최근에는 네트워크 구조 탐색 방법인 NAS와 같은 자동 모델 탐색 방법을 활용하기도 한다.